Parallel RooFit and Interface Improvements

Patrick Bos (Netherlands eScience Center),
Wouter Verkerke (ATLAS/Nikhef), Zef Wolffs (ATLAS/Nikhef)

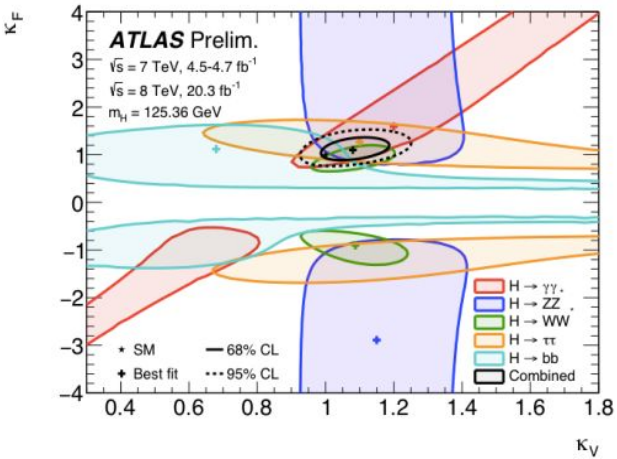
Introduction

- Original RooFit implements very simple parallel strategy
  - Split calculation of each likelihood call in N equal pieces
  - Load balancing scales poorly for workspaces with many component likelihoods of different sizes

- A new initiative to parallelize RooFit started ± 4 years ago [1]
  - Parallelize at level of gradient calculations, rather than at level of likelihood evaluation
  - This new strategy improves load balancing

- Also overhaul of both internal and user interface classes for likelihood component calculations

- Infrastructure available in ROOT 6.26, large scale testing currently undergoing with prospective to connect to public interfaces in next release



Example Higgs combination fit result, these fits currently easily require many hours to complete

[1] Bos, E. G. P., Burgard, C. D., Croft, V. A., Hageboeck, S., Moneta, L., Pelupessy, I., ... & Verkerke, W. (2020). Faster RooFitting: Automated parallel calculation of … (see last slide)

Nikhef

Parallelization Strategy

Parallelization Strategy

- Serial MIGRAD minimization for likelihood with N parameters

$$\frac{f(x+h)-f(x)}{h}$$

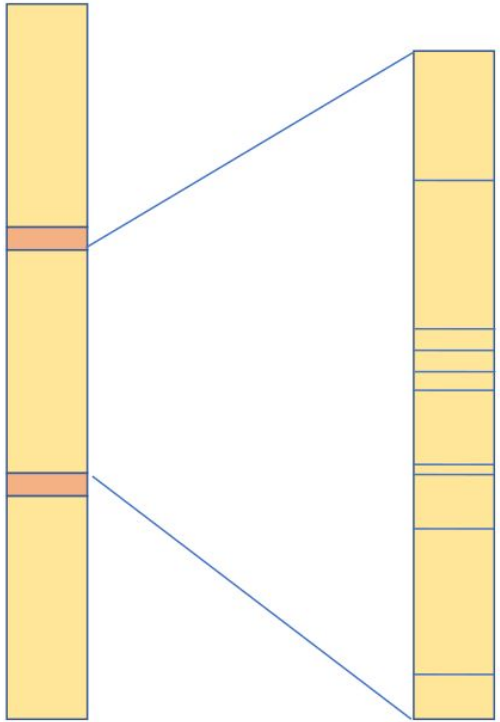
N x gradient calculation
= 2N likelihood evals

line search (O(3) evals)

N x gradient calculation
= 2N likelihood evals

line search (O(3) evals)

N x gradient calculation
= 2N likelihood evals

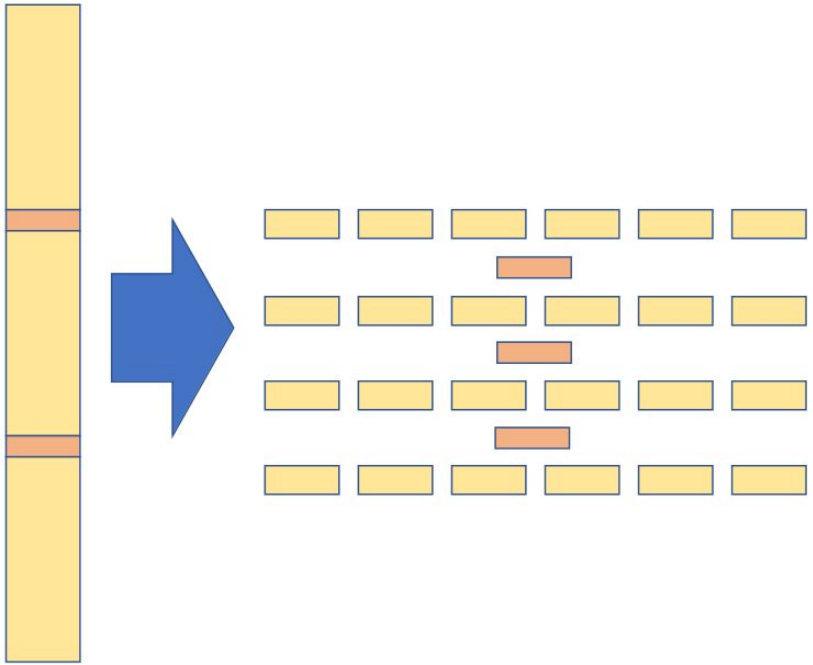


NB: CPU time for gradient varies strongly with parameter:

- some parameters only appear in subset of likelihood components
- complexity of component likelihood calculation varies
- effect of optimization varies

Parallelization Strategy

- Parallelize RooFit/MIGRAD minimization



- In parallel gradient calculation part dynamic load balancing over workers through random work stealing algorithm
- Designed to have maximum speed impact of complex fits with many parameters
- For N=O(100), typically only 2% of CPU time spent in line-search so serial line search step has limited impact on CPU scaling
- Worked closely with ROOT team for Minuit interface and validation with special attention to obtain exact or as close-as-possible to identical results (within num precision)

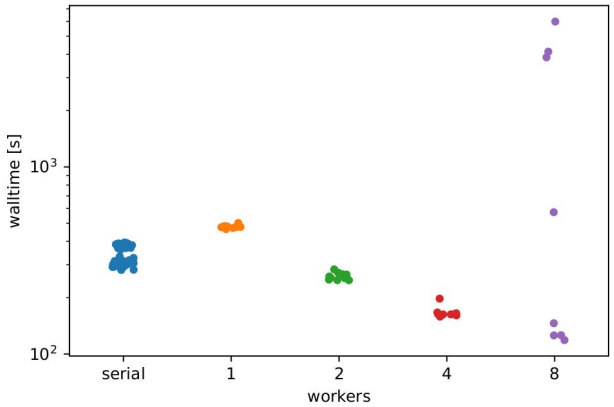
Nikhef

Benchmarking

Current Status & Benchmarking

- Now doing scaling tests for real ATLAS workspaces
  - First candidate is ATLAS H->WW fit (~5 min fit time currently), see results below
  - Next is Higgs combination fit (~2 hours fit time currently). This is work in progress

- When scaling tests and benchmarking are done, the plan is to ship the built tools with ROOT by default so users can also analyze their own code and workspaces for potential bottlenecks

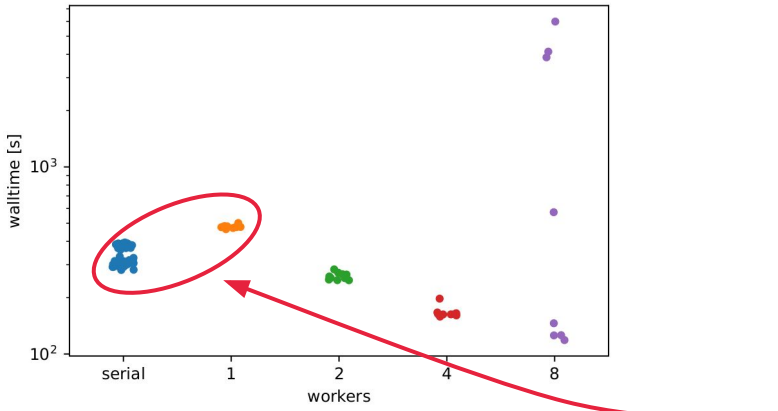


H->WW workspace fit times with increasing numbers of workers

Current Status & Benchmarking

- Now doing scaling tests for real ATLAS workspaces
  - First candidate is ATLAS H->WW fit (~5 min fit time currently), see results below
  - Next is Higgs combination fit (~2 hours fit time currently). This is work in progress

- When scaling tests and benchmarking are done, the plan is to ship the built tools with ROOT by default so users can also analyze their own code and workspaces for potential bottlenecks



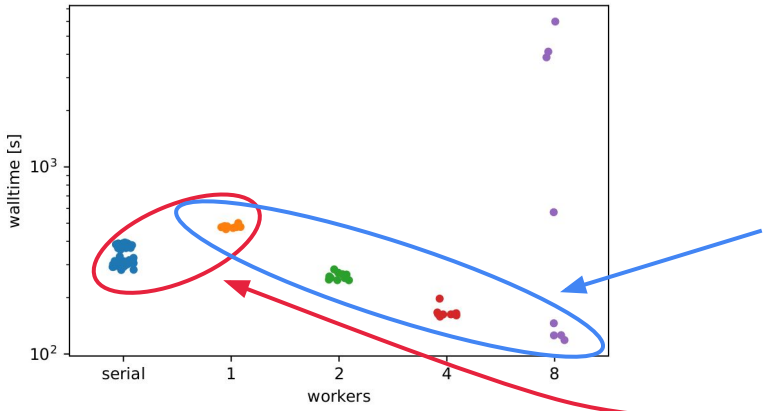
H->WW workspace fit times with increasing numbers of workers

Slight overhead in going from serial (current) implementation to parallel, this is to be expected

Current Status & Benchmarking

- Now doing scaling tests for real ATLAS workspaces
  - First candidate is ATLAS H->WW fit (~5 min fit time currently), see results below
  - Next is Higgs combination fit (~2 hours fit time currently). This is work in progress

- When scaling tests and benchmarking are done, the plan is to ship the built tools with ROOT by default so users can also analyze their own code and workspaces for potential bottlenecks



H->WW workspace fit times with increasing numbers of workers

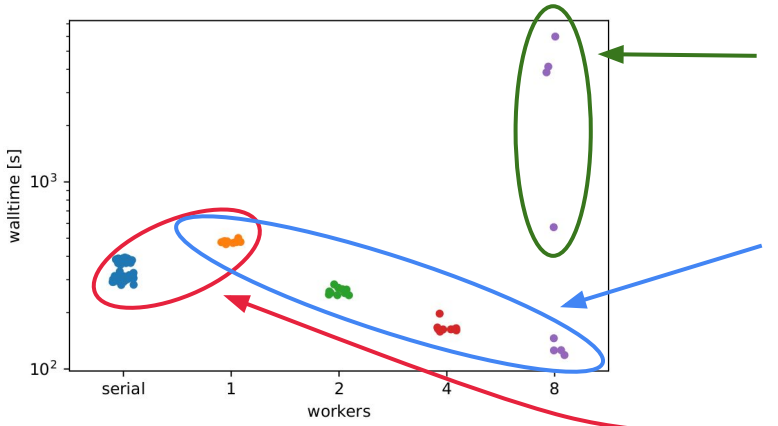
Good scaling behaviour up to 8 workers, with some runs also scaling well for 8 workers.

Slight overhead in going from serial (current) implementation to parallel, this is to be expected

Current Status & Benchmarking

- Now doing scaling tests for real ATLAS workspaces
  - First candidate is ATLAS H->WW fit (~5 min fit time currently), see results below
  - Next is Higgs combination fit (~2 hours fit time currently). This is work in progress

- When scaling tests and benchmarking are done, the plan is to ship the built tools with ROOT by default so users can also analyze their own code and workspaces for potential bottlenecks



H->WW workspace fit times with increasing numbers of workers

Some runs significantly slower for 8 workers, under investigation! Perhaps something to do with worker-queue process communication, or race conditions. Once this is solved, scale up to 64 processes

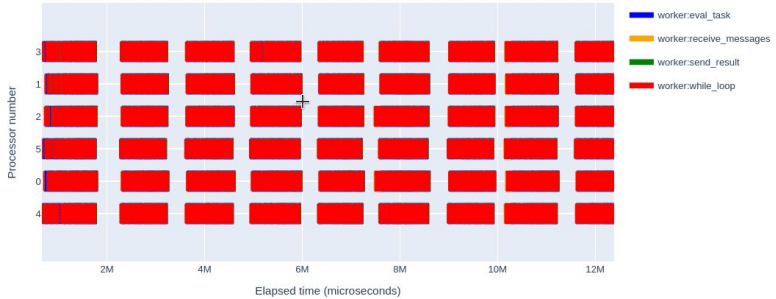
Good scaling behaviour up to 8 workers, with some runs also scaling well for 8 workers.

Slight overhead in going from serial (current) implementation to parallel, this is to be expected

Current Status & Benchmarking

- Created multiprocess timer that can keep track of each process' task within the parallel framework.
  - Could be built into ROOT by default in the future

- Preparing more detailed analysis of time expenditures - what is fraction of time spent in each worker and master process in calculations, communication, other overhead.
  - This will help identify and eliminate scaling bottlenecks, for example those shown in the previous slide

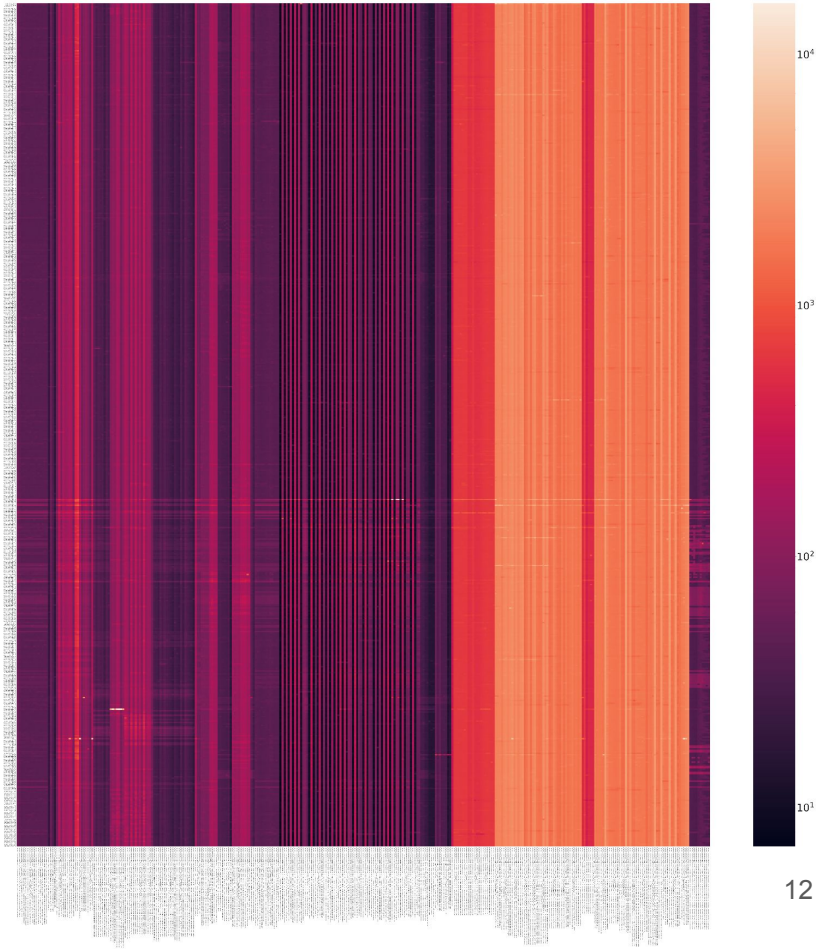


First profiling results of H->WW workspace fit

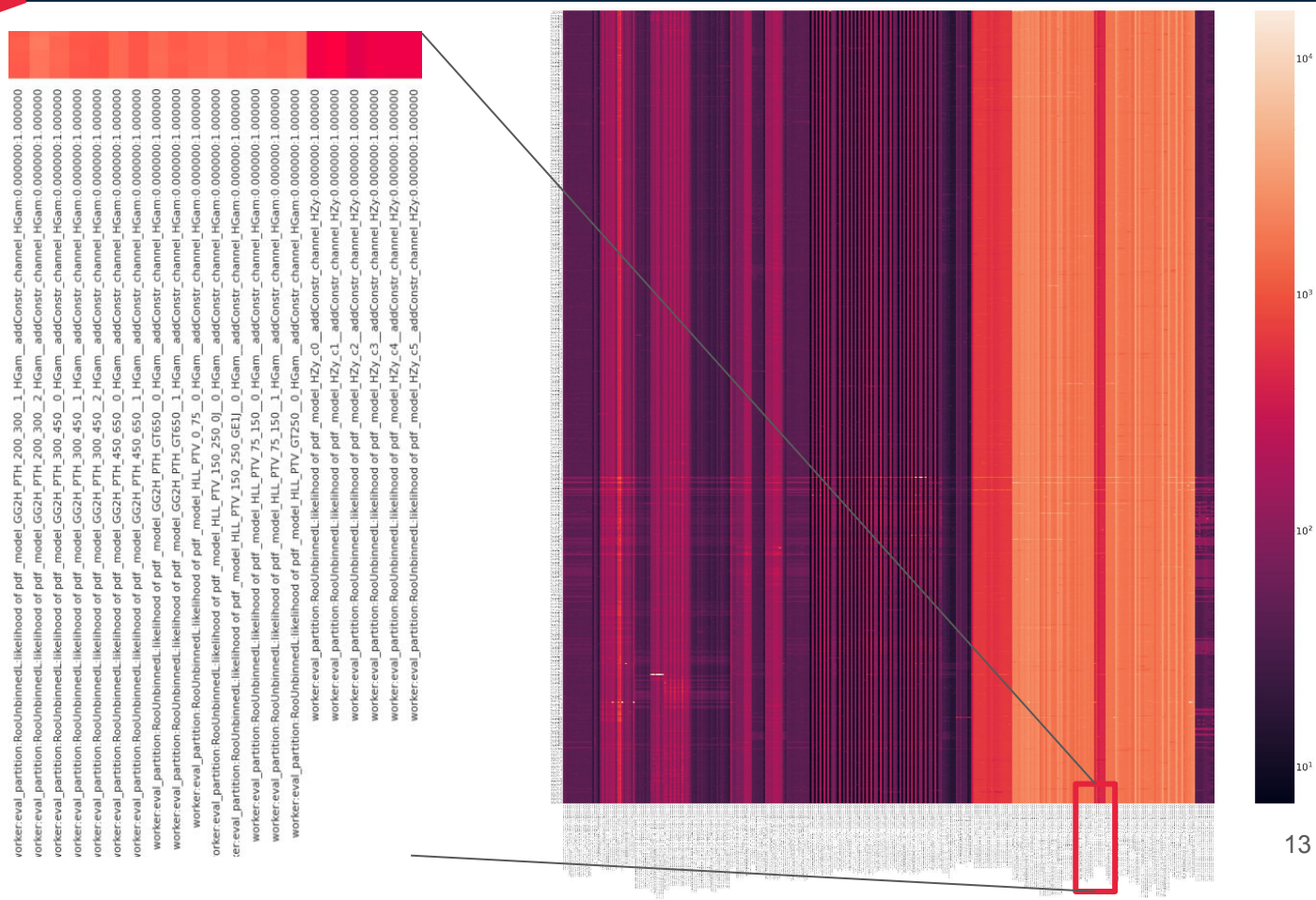
Current Status & Benchmarking

- Currently we are working to map out the entire Higgs combination fit using the previously introduced multiprocess timer, this includes timing all likelihood evaluations per partial derivative

- In the future, this could be shipped with ROOT, allowing the user to scrutinize their own workspaces for potential bottlenecks
    - If you're interested in joining as an alpha tester (so that we can try to speed up your workspaces), contact us!



Current Status & Benchmarking



Interface Developments

Interface Developments

- In order to hide the added complexities from the user, a new minimization and likelihood building interface was developed
  - This allows the user to be agnostic towards the used computational strategy and rely on RooFit to make the optimal choice in the back-end
  - New computational strategies (e.g. GPU, analytical derivation, these were mentioned Jonas' talk!) can simply be "plugged in" to the interface

- Newly developed classes and functions are namespaced clearly
  - Multiprocess implementation under `RooFit::MultiProcess`
  - Likelihood building implementation under `RooFit::TestStatistics`

- Enumerations are used to allow the user to specify options in calls to functions or class constructors

```
RooMinimizer m(likelihood, LikelihoodMode::serial, LikelihoodGradientMode::multiprocess);
```

Interface Developments

Specify namespaces as per previous slide

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;

void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Interface Developments

Specify namespaces as per previous slide

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;
```

Specify number of workers to use

```
void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Interface Developments

Specify namespaces as per previous slide

Specify number of workers to use

Create or import a workspace

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;

void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Interface Developments

Specify namespaces as per previous slide

Specify number of workers to use

Create or import a workspace

Build a likelihood, note that there is no need to specify the type. RooFit chooses the appropriate likelihood based on the input

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;

void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Interface Developments

Specify namespaces as per previous slide

Specify number of workers to use

Create or import a workspace

Build a likelihood, note that there is no need to specify the type. RooFit chooses the appropriate likelihood based on the input

Create the minimizer and optionally specify a backend

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;

void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Interface Developments

Specify namespaces as per previous slide

Specify number of workers to use

Create or import a workspace

Build a likelihood, note that there is no need to specify the type. RooFit chooses the appropriate likelihood based on the input

Create the minimizer and optionally specify a backend

Minimize!

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;

void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Interface Developments

Specify namespaces as per previous slide

Specify number of workers to use

Create or import a workspace

Build a likelihood, note that there is no need to specify the type. RooFit chooses the appropriate likelihood based on the input

Create the minimizer and optionally specify a backend

Minimize!

```
using namespace RooFit::MultiProcess;
using namespace RooFit::TestStatistics;

void demo()
{
    Config::setDefaultNWorkers(2);

    RooWorkspace w = RooWorkspace();
    w.factory("Gaussian::g(x[-5,5],mu[0,-3,3],sigma[1])");
    RooAbsPdf *pdf = w.pdf("g");
    RooDataSet *data = pdf->generate(RooArgSet(*w.var("x")), 10000);
    RooRealVar *mu = w.var("mu");

    std::shared_ptr<RooAbsL> likelihood = buildLikelihood(pdf, data);

    RooMinimizer m(likelihood, LikelihoodMode::serial,
                   LikelihoodGradientMode::multiprocess);

    m.migrad();
}
```

Conclusion

- A new parallel implementation of RooFit was developed that parallelizes at the level of gradient calculations
  - Scales well through dynamic load-balancing

- An interface was added to RooFit as well to hide the added complexity in the background, and allow future backend computational strategies to be used through the same interface as well

- Scaling tests and benchmarking underway, new implementation looks promising but final bottlenecks still need to be eliminated
  - Multiprocess timer and visualization scripts may allow the user to undertake scaling studies for their particular minimization problems in the future

References

[1] Bos, E. G. P., Burgard, C. D., Croft, V. A., Hageboeck, S., Moneta, L., Pelupessy, I., ... & Verkerke, W. (2020). Faster RooFitting: Automated parallel calculation of collaborative statistical models. In *EPJ Web of Conferences* (Vol. 245, p. 06027). EDP Sciences.

Nikhef

Backup

Parallel RooFit Benchmarking - Extra background

- Their solution: Gradient-based parallelisation
  - Take the likelihood $\ell(\theta\,;\mathbf{y}) = \ln L_n(\theta\,;\mathbf{y})$ and parallelise the numerical minimization of it at the gradient calculation level:

$$\frac{\partial \ell}{\partial \theta_1} = 0, \quad \frac{\partial \ell}{\partial \theta_2} = 0, \quad \ldots, \quad \frac{\partial \ell}{\partial \theta_k} = 0,$$

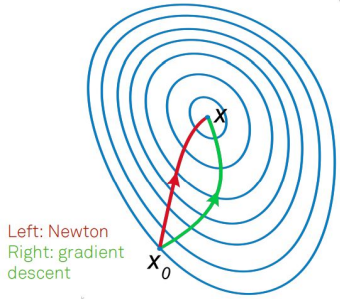
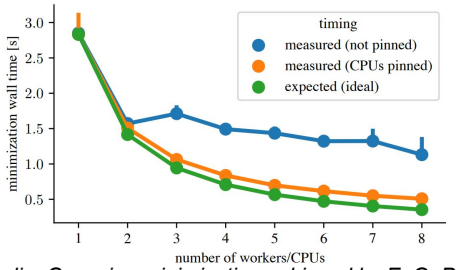


Illustration of two numerical minimization methods

RooFit uses Minuit, a Quasi-Newton minimization method to minimize likelihood w.r.t. parameters, which iteratively proceeds to a minimum and on every iteration calculates above derivatives for all parameters -> Many calculations!

- Previous work succeeded in making minimization a lot faster, however they also observed some issues with particular types of fits

- My Goal: Make sure that this development also benefits ATLAS, and that we can make compute-heavy fits that are done within ATLAS (such as Higgs combination fits) also a lot faster.



Speedup of N-dim Gaussian minimization achieved by E. G. P. Bos et al [1].